SAMEER MOTWANI
CS 6240-01
Homework 3

DESIGN DISCUSSION

Preprocessing

Parsing Job Pseudo code

```
class IpParserMapper {
        map(key, value){
                parses the input files in the for of "pageName->(list of all outlinks)->pageRank"
        }
```

Page Rank Iterator Psuedo Code

```
//mapper
class PageRankMapper(Object, Text, Text, OutputData){
    setup(){
        itr = get from Configuration;
        totalPages = get from Configuration;
    }
    map(key, value){
        line = value.toString();
        lineSplit[] = line.split("->");
        outlinks[] = lineSplit[1]; //getting the list of outlinks
        if(itr == 0){
            pageRval = 1 / totalPages;
        }
        else
            pageRval = lineSplit[2];
        if(outlinks == null){
            set the Global counter "danglingFactor"
        }
        else{
            write(pageName, new Output(true, pageRval, outlinks.length))
        }
        write (lineSplit[0], new Output(false, lineSplit[1]);
    }


//reducer
class PageRankReducer(Text, OutputData, Text, NullWritable){
        setup(){
            danglingFactor = get from Config
                    pageCOunt = get from Config
                    pageList = new HashSet<String>
        }
        reduce(key, Iterable<OutputData>){
            for Output d : values {
                if(d.isnotPRdata)
                    outlinklist = d.getOutlinklist();
                else{
                    summation += d.getProfm / d.getOutlinkCount;
                }
                randomjump = alpha / pageCount;
                followingLink = (1.0 - alpha) * ((danglingFactor / pageCount) + summation)
                        pageRank = randomjump + followingLink
                        String output = "pagename->(list)->pageRank"
                        write(output, Nullwritable.get())
                        cleanup()
                        { update noOfPage counter based on pageList count}
            }
```

Page Rank Top 100 Psuedo Code

```
// mapper
        class OutputMapper(Object, Text, DoubleWritable, Text){
                Map<String, Double> topPages; // hashmap to store top pages
                setup(){
                        topPages = HashMap
                }
                map(){
                        parts = value.split("->")
                                        put in hashmap pageRank and pageName;
                }
                cleanup(){
                        sort in descending order of values in hashmap
                        write(pagerank, page)
                }
        }

// reducer
        class OutputReducer(DoubleWritable, Text, DoubleWritable, Text){
                        TreeMap repToRecordMap = new TreeMap
                        reduce(Text page_name, Iteratable<Text> value) {
                repToRecordMap.put(value.getPageRank(), page_name)
                if(repToRecordMap.size() > 100) {
                        repToRecordMap.removeTheFirstValue()
                }
        }
        cleanup() {
                for every record in repToRecordMap in descending order {
                        emit(record.value, record.key)
                }
        }
}
```

Report the amount of data transferred from Mappers to Reducers, and from Reducers to HDFS, in each iteration of the PageRank computation. Does it change over time?

Iterations	Map O/p Bytes	Map O/p Materialized Bytes	Reducer to HDFS bytes
1	3937896702	1371669557	1418672042
2	3947664654	1756546764	1418517639
3	3947664654	1757192095	1418486319
4	3947664654	1757116864	1418487655
5	3947664654	1757085181	1418470552
6	3947664654	1757046124	1418469916
7	3947664654	1757007989	1418461086
8	3947664654	1756883433	1418456160
9	3947664654	1756842101	1418463494
10	3947664654	1756778381	1418449331

As we can see there is a slight increase in Map O/P bytes from iteration1 to iteration 2 and then the value remains the same, this can be due to the fact that the dangling nodes being handled.

Performance Comparison

Report for both configurations (i) pre-processing time, (ii) time to run ten iterations of PageRank, and (iii) time to find the top-100 pages. There should be 2*3=6 time values.

Time comparison of 2 EMR runs:

Number of AWS Machines	Job – Parser(mins)	Job- Page Rank Calculate(mins)	Job- Top 100(mins)
6	24	25.5	1.1
11	11	17.9	0.733

Critically evaluate the runtime results by comparing them against what you had expected to see and discuss your findings. Make sure you address the following question: Which of the computation phases showed a good speedup? If a phase seems to show fairly poor speedup, briefly discuss possible reasons—make sure you provide concrete evidence, e.g., numbers from the log file or analytical arguments based on the algorithm's properties

Answer:

Speedup for different phases is as below:

Preprocessing: time on 6 machines/time on 11 machines= 24/11=2.18
In preprocessing, we are getting speedup of 2.18 which indicates great parallelism.

Page rank calculation: 25.5/17.9=1.424
In page-rank, we are getting a speedup of 1.424 which indicates good degree of parallelism.

Top 100 pages: 1.1/0.733=1.5
In Top 100 page calculation, we are getting speedup of 1.5 which indicates good degree of parallelism.

All the phases have shown a good degree of parallelism.

Report the top-100 Wikipedia pages with the highest Page Ranks, along with their rank values and sorted from highest to lowest, for both the simple and full datasets. Do they seem reasonable based on your intuition about important information on Wikipedia?

Answer:

1. The final result output files are present in the output folder.
2. The output of both the files look reasonable and as expected as we can see that the country names like United States, England, India, Germany appear in the top 100 as we can assume these are most commonly used in all the wiki pages.